**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Волков М. А.

Группа: М80-207Б-19

Преподаватель: Чернышев Л. Н.

Дата:

Оценка:

1. **Постановка задачи**

Спроектировать простейший «графический» векторный редактор.

Требование к функционалу редактора:

• создание нового документа

• импорт документа из файла

• экспорт документа в файл

• создание графического примитива (согласно варианту задания)

• удаление графического примитива

• отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout)

• реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

• Создание графических примитивов необходимо вынести в отдельный класс – Factory.

• Сделать упор на использовании полиморфизма при работе с фигурами;

• Взаимодействие с пользователем (ввод команд) реализовать в функции main;

*Вариант: Прямоугольник, трапеция, ромб*

1. **Описание программы**

Задание разбито на несколько файлов: *main.cpp, document.h, factory.h, Rectangle.h, Rhombus.h, Trapeze.h, Figures.h*

Файлы *Rectangle.h, Rhombus.h, Trapeze.h, Figures.h* были взяты из лабораторной работы 3 и были доработаны. Я сделал так потому, что распределение вариантов предполагает дорабатывать свою предыдущую программу. Доработка заключается в том, что теперь все классы шаблонные. Из-за этого реализация методов ранее вынесенная в файлы .cpp теперь находится в .h. Также для трапеции был использован *std::enable\_if.* Это сделано потому, что вычисление координат происходит там путем взятия корня и деления на переменную типа double. Поэтому вычисление координат будет происходить только в double. В свою очередь, если все фигуры будут типа int, то программа не сломается, а будет писать, что для трапеции она не может вывести координаты, так как фигура типа int.

В файле *factory.h* написана реализация шаблонного класса TFactory, которая отвечает за создание фигур и запись этих фигур в файл с последующей загрузкой из него.

В свою очередь в файле *document.h* реализован шаблонный класс TDocument, в котором происходит вся основная «магия». В этом классе реализованы все методы работы с файлами. В том числе и метод *Undo,* который, благодаря записи всех действий в стек, может вернуть действия, сделанные пользователем с начала старта программы.

1. **Наборы и результаты выполнения тестов**

*Тест 1: используется тип double*

> h

> 'n' - create new document

> 'o' - open document

> 's' - save document

> '+' - add a figure

when you add figure:

first - enter your position you want insert your figure

second - enter figure ID:

1 - rhombus

2 - rectangle

3 - trapeze

> '-' - remove a figure

when you remove figure, enter your position you want to delete

> 'p' - print document

> 'u' - undo changes

> 'h' - show this message

> 'e' - exit

> + 0 3

Enter trapeze as follows: x y a b c

x, y is a left bottom point cords

a, b and c are larger, smaller base and height

0 0 25 5 4

> p

Printing document:

[1] Trapeze vertices: [(0, 0), (10, 9.16515), (15, 9.16515), (25, 0)]

> s

log.bin

Saved document to log.bin

> - 0

> p

Printing document:

> o lob.bin

No such file in directory

> o

log.bin

Loaded document from log.bin

> p

Printing document:

[1] Trapeze vertices: [(0, 0), (10, 9.16515), (15, 9.16515), (25, 0)]

> u

Nothing to undo!

> e

Process finished with exit code 0

*Тест 2: используется тип int*

> h

> 'n' - create new document

> 'o' - open document

> 's' - save document

> '+' - add a figure

when you add figure:

first - enter your position you want insert your figure

second - enter figure ID:

1 - rhombus

2 - rectangle

3 - trapeze

> '-' - remove a figure

when you remove figure, enter your position you want to delete

> 'p' - print document

> 'u' - undo changes

> 'h' - show this message

> 'e' - exit

> + 0 3

Enter trapeze as follows: x y a b c

x, y is a left bottom point cords

a, b and c are larger, smaller base and height

0 0 25 3 1

> p

Printing document:

[1] Trapeze: [ I cant calculate coordinates, because the type is int ]

> e

Process finished with exit code 0

1. **Листинг программы**

**main.cpp**

/\*  
 \* Волков Матвей Андреевич  
Спроектировать простейший «графический» векторный редактор.  
Требование к функционалу редактора:  
• создание нового документа  
• импорт документа из файла  
• экспорт документа в файл  
• создание графического примитива (согласно варианту задания)  
• удаление графического примитива  
• отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout)  
• реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.  
  
Требования к реализации:  
• Создание графических примитивов необходимо вынести в отдельный класс – Factory.  
• Сделать упор на использовании полиморфизма при работе с фигурами;  
• Взаимодействие с пользователем (ввод команд) реализовать в функции main;  
Вариант: Прямоугольник, трапеция, ромб  
\*/  
  
#include "document.h"  
  
//нужно, чтобы было 2 одинаковых типа  
#define yourTYPE int, int  
  
int main() {  
 std::ofstream fout;  
 std::ifstream fin;  
  
 TDocument<yourTYPE> doc;  
 std::string s;  
 while ((std::cout << "> ") && (std::cin >> s)) {  
 if (s == "n") {  
 doc.CreateNew();  
 std::cout << "Created new document" << std::endl;  
 }  
 else if (s == "o") {  
 std::cin >> s;  
 fin.open(s, std::ios::in | std::ios::binary);  
 if (!fin.is\_open()) {  
 std::cout << "No such file in directory" << std::endl;  
 fin.close();  
 }  
 else {  
 doc.LoadFromFile(fin);  
 std::cout << "Loaded document from " << s << std::endl;  
 fin.close();  
 }  
 }  
 else if (s == "s") {  
 std::cin >> s;  
 fout.open(s, std::ios::out | std::ios::binary | std::ios::trunc);  
 if (!fout.is\_open()) {  
 std::cout << "Cannot open this file" << std::endl;  
 }  
 else {  
 doc.SaveToFile(fout);  
 std::cout << "Saved document to " << s << std::endl;  
 fout.close();  
 }  
 }  
 else if (s == "+") {  
 size\_t pos;  
 unsigned short type;  
 std::cin >> pos >> type;  
 doc.Add(pos, type);  
 }  
 else if (s == "-") {  
 size\_t pos;  
 std::cin >> pos;  
 doc.Delete(pos);  
 }  
 else if (s == "p") {  
 std::cout << "Printing document:" << std::endl;  
 std::cout << doc;  
 }  
 else if (s == "u") {  
 doc.Undo();  
 }  
 else if (s == "h") {  
 std::cout << "> \'n\' - create new document" << std::endl;  
 std::cout << "> \'o\' - open document" << std::endl;  
 std::cout << "> \'s\' - save document" << std::endl;  
 std::cout << "> \'+\' - add a figure\n"  
 "when you add figure:\nfirst - enter your position you want insert your figure\n"  
 "second - enter figure ID:\n"  
 " 1 - rhombus\n 2 - rectangle\n 3 - trapeze" << std::endl;  
 std::cout << "> \'-\' - remove a figure\n"  
 "when you remove figure, enter your position you want to delete" << std::endl;  
 std::cout << "> \'p\' - print document" << std::endl;  
 std::cout << "> \'u\' - undo changes" << std::endl;  
 std::cout << "> \'h\' - show this message" << std::endl;  
 std::cout << "> \'e\' - exit\n" << std::endl;  
 }  
  
 else if(s == "e"){  
 return 0;  
 }  
 else {  
 std::cout << "Unknown command. Type \'h\' to show help" << std::endl;  
 }  
 }  
 return 0;  
}

**document.h**

#ifndef OOP\_EXERCISE\_07\_DOCUMENT\_H  
#define OOP\_EXERCISE\_07\_DOCUMENT\_H  
  
//тип, используемый вами  
#define tempTYPE class Tpoint, class Tside  
  
//если нет типа у глобального класса фигуы, нужно закомментировать  
//если имеется ваш собственный тип, обязательно нужно писать <>  
#define FIGURE\_TYPE <Tpoint>  
  
//здесь находится применение ваших типов  
#define TYPES Tpoint, Tside  
  
#include <list>  
#include <stack>  
  
#include "factory.h"  
  
template<tempTYPE>  
class TDocument {  
private:  
 struct IAction;  
  
 using figure\_pointer = std::shared\_ptr<Figures FIGURE\_TYPE>;  
 using action\_pointer = std::shared\_ptr<IAction>;  
 using const\_iterator = typename std::list< figure\_pointer >::const\_iterator;  
  
 std::list< figure\_pointer > FiguresList;  
 std::stack< action\_pointer > ActionStack;  
  
 struct IAction {  
 virtual void PerformAction(TDocument & fact) = 0;  
 virtual ~IAction() {}  
 };  
  
 class TDeleteAction : public IAction {  
 private:  
 size\_t DeletePos;  
 public:  
 TDeleteAction(const size\_t & pos) : DeletePos(pos) {}  
 void PerformAction(TDocument & fact) override {  
 fact.Delete(DeletePos);  
 }  
 };  
  
 class TAddAction : public IAction {  
 private:  
 size\_t AddPos;  
 figure\_pointer AddFigure;  
 public:  
 TAddAction(const size\_t & pos, const figure\_pointer & fig) : AddPos(pos), AddFigure(fig) {}  
 void PerformAction(TDocument & fact) override {  
 fact.AddFigure(AddPos, AddFigure);  
 }  
 };  
  
  
public:  
 void CreateNew() {  
 while (!ActionStack.empty()) {  
 ActionStack.pop();  
 }  
 FiguresList.clear();  
 }  
  
 void LoadFromFile(std::ifstream& in) {  
 CreateNew();  
 size\_t n;  
 in.read((char\*) &(n), sizeof (size\_t));  
 for (size\_t i = 0; i < n; ++i) {  
 unsigned int type;  
 in.read((char\*) &(type), sizeof(unsigned));  
 if (type == RHOMBUS\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Rhombus<TYPES>>::Read(in));  
 } else if (type == RECTANGLE\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Rectangle<TYPES>>::Read(in));  
 } else if (type == TRAPEZE\_TYPE\_ID) {  
 FiguresList.push\_back(TFactory< TYPES, Trapeze<TYPES>>::Read(in));  
 }  
 }  
 }  
  
 void SaveToFile(std::ofstream& out) {  
 size\_t n = FiguresList.size();  
 out.write((const char\*) &n, sizeof (size\_t));  
 for (const\_iterator it = FiguresList.begin(); it != FiguresList.end(); ++it) {  
 (\*it)->Write(out);  
 }  
 }  
  
 void Add(const size\_t& pos, const unsigned int& figureID) {  
 if (figureID == RHOMBUS\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Rhombus<TYPES> >::CreateFigure());  
 } else if (figureID == RECTANGLE\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Rectangle<TYPES> >::CreateFigure());  
 } else if (figureID == TRAPEZE\_TYPE\_ID) {  
 AddFigure(pos, TFactory<TYPES, Trapeze<TYPES> >::CreateFigure());  
 }  
 }  
  
 void AddFigure(const size\_t & pos, const figure\_pointer & fig) {  
 if (pos > FiguresList.size()) {  
 FiguresList.push\_back(fig);  
 auto delAct = new TDeleteAction(FiguresList.size());  
 ActionStack.push(action\_pointer(delAct));  
 } else {  
 size\_t cur = 0;  
 const\_iterator it = FiguresList.begin();  
 while (cur < pos) {  
 ++cur;  
 ++it;  
 }  
 FiguresList.insert(it, fig);  
 auto delAct = new TDeleteAction(pos + 1);  
 ActionStack.push(action\_pointer(delAct));  
 }  
 }  
  
 void Delete(const size\_t & pos) {  
 if (FiguresList.empty()) {  
 std::cout << "Nothing to delete!" << std::endl;  
 return;  
 }  
 if (pos > FiguresList.size()) {  
 auto addAct = new TAddAction(FiguresList.size() - 1, FiguresList.back());  
 ActionStack.push(action\_pointer(addAct));  
 FiguresList.pop\_back();  
 } else {  
 size\_t cur = 1;  
 const\_iterator it = FiguresList.begin();  
 while (cur < pos) {  
 ++cur;  
 ++it;  
 }  
 auto addAct = new TAddAction(cur - 1, \*it);  
 ActionStack.push(action\_pointer(addAct));  
 FiguresList.erase(it);  
 }  
 }  
  
 void Undo() {  
 if (ActionStack.empty()) {  
 std::cout << "Nothing to undo!" << std::endl;  
 } else {  
 ActionStack.top()->PerformAction(\*this);  
 ActionStack.pop();  
 ActionStack.pop();  
 }  
 }  
  
  
 friend std::ostream & operator << (std::ostream & of, const TDocument<TYPES> & fact) {  
 TDocument::const\_iterator it = fact.FiguresList.begin();  
 for (size\_t i = 0; i < fact.FiguresList.size(); ++i) {  
 of << "[" << i + 1 << "] ";  
 (\*it)->Print(of);  
 of << std::endl;  
 ++it;  
 }  
 return of;  
 }  
};  
  
#endif //OOP\_EXERCISE\_07\_DOCUMENT\_H

**factory.h**

#ifndef OOP\_EXERCISE\_07\_FACTORY\_H  
#define OOP\_EXERCISE\_07\_FACTORY\_H  
  
#include <memory>  
  
#include "Rectangle.h"  
#include "Rhombus.h"  
#include "Trapeze.h"  
  
template<class Tpoint,class Tside, class FIGURE>  
class TFactory;  
  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Rectangle<Tpoint, Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curHeight, curWidth;  
 std::cout << "Enter rectangle as follows: x y a b" << std::endl;  
 std::cout << "x, y is a left bottom point cords" << std::endl;  
 std::cout << "a and b are width and heigth" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curHeight >> curWidth;  
 auto rect = new Rectangle<Tpoint,Tside>(curCords, curHeight, curWidth);  
 return std::shared\_ptr<Figures<Tpoint>>(rect);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& file) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curHeight, curWidth;  
 file.read((char\*) &(curCords.first), sizeof(Tpoint));  
 file.read((char\*) &(curCords.second), sizeof (Tpoint));  
 file.read((char\*) &(curHeight), sizeof (Tside));  
 file.read((char\*) &(curWidth), sizeof (Tside));  
 auto rect = new Rectangle<Tpoint,Tside>(curCords, curHeight, curWidth);  
 return std::shared\_ptr<Figures<Tpoint>>(rect);  
 }  
};  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Trapeze<Tpoint,Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curGreaterBase, curSmallerBase, curHeight;  
 std::cout << "Enter trapeze as follows: x y a b c" << std::endl  
 << "x, y is a left bottom point cords" << std::endl  
 << "a, b and c are larger, smaller base and height" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curGreaterBase >> curSmallerBase >> curHeight;  
 auto trap = new Trapeze<Tpoint,Tside>(curCords, curGreaterBase, curSmallerBase, curHeight);  
 return std::shared\_ptr<Figures<Tpoint>>(trap);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& in) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curGreaterBase, curSmallerBase, curHeight;  
 in.read((char\*) &(curCords.first), sizeof (Tpoint));  
 in.read((char\*) &(curCords.second), sizeof (Tpoint));  
 in.read((char\*) &(curGreaterBase), sizeof (Tside));  
 in.read((char\*) &(curSmallerBase), sizeof (Tside));  
 in.read((char\*) &(curHeight), sizeof (Tside));  
 auto trap = new Trapeze<Tpoint,Tside>(curCords, curGreaterBase, curSmallerBase, curHeight);  
 return std::shared\_ptr<Figures<Tpoint>>(trap);  
 }  
};  
  
  
template<class Tpoint, class Tside>  
class TFactory< Tpoint, Tside, Rhombus<Tpoint,Tside> > {  
public:  
 static std::shared\_ptr<Figures<Tpoint>> CreateFigure() {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curD1, curD2;  
 std::cout << "Enter rhombus as follows: x y d1 d2" << std::endl  
 << "x, y is a left bottom point cords" << std::endl  
 << "d1 and d2 are diagonals of rhombus" << std::endl;  
 std::cin >> curCords.first >> curCords.second >> curD1 >> curD2;  
 auto pRhombus = new Rhombus<Tpoint,Tside>(curCords, curD1, curD2);  
 return std::shared\_ptr<Figures<Tpoint>>(pRhombus);  
 }  
  
 static std::shared\_ptr<Figures<Tpoint>> Read(std::ifstream& in) {  
 std::pair<Tpoint, Tpoint> curCords;  
 Tside curD1, curD2;  
 in.read((char\*) &(curCords.first), sizeof (Tpoint));  
 in.read((char\*) &(curCords.second), sizeof (Tpoint));  
 in.read((char\*) &(curD1), sizeof (Tside));  
 in.read((char\*) &(curD2), sizeof (Tside));  
 auto pRhombus = new Rhombus<Tpoint,Tside>(curCords, curD1, curD2);  
 return std::shared\_ptr<Figures<Tpoint>>(pRhombus);  
 }  
};  
  
#endif //OOP\_EXERCISE\_07\_FACTORY\_H

**Figures.h**

#ifndef OOP\_EXERSICE\_03\_FIGURES\_H  
#define OOP\_EXERSICE\_03\_FIGURES\_H  
#include <fstream>  
#include <iostream>  
#include <cmath>  
  
template<class T1,class T2>  
struct is\_int : std::false\_type {};  
template<>  
struct is\_int<int,int>:std::true\_type {};  
  
template<class T1,class T2>  
struct is\_double : std::false\_type {};  
template<>  
struct is\_double<double,double>:std::true\_type {};  
  
template<class T>  
class Figures {  
protected:  
 using coord = std::pair<T,T>;  
 coord point;  
  
public:  
 virtual void Print(std::ostream& out) = 0;  
 virtual void Write(std::ofstream& file){};  
 virtual ~Figures() = default;  
};  
  
template<class T1, class T2>  
inline std::ostream & operator<<(std::ostream& out, const std::pair<T1, T2>& pair){  
 out << "(" << pair.first << ", " << pair.second << ")";  
 return out;  
}  
  
#endif //OOP\_EXERSICE\_03\_FIGURES\_H

**Rectangle.h**

#ifndef RECTANGLE\_HPP  
#define RECTANGLE\_HPP  
#include "Figures.h"  
  
const unsigned int RECTANGLE\_TYPE\_ID = 2;  
  
template<class T1, class T2>  
class Rectangle : public Figures<T1> {  
private:  
 T2 A, B;  
public:  
 Rectangle(std::pair<T1,T2> \_point, const T2 &a, const T2 &b): A(a), B(b){// x and y are left-bottom point of figure  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (A < 0 or B < 0) {  
 throw std::invalid\_argument("Invalid rectangle parameters!");  
 }  
 }  
  
  
 void Print(std::ostream& out) override {  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(RECTANGLE\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (T1));  
 file.write((const char\*) &(this->point.second), sizeof(T1));  
 file.write((const char\*) &(this->A), sizeof(T2));  
 file.write((const char\*) &(this->B), sizeof (T2));  
 }  
  
 template<class U>  
 friend std::ostream & operator << (std::ostream & out, const Rectangle<U,U> & rectangle){  
 out << "Rectangle vertices: [";  
 out << std::pair<U,U> (rectangle.point.first, rectangle.point.second) << ", ";  
 out << std::pair<U,U> (rectangle.point.first, rectangle.point.second + rectangle.B) << ", ";  
 out << std::pair<U,U> (rectangle.point.first + rectangle.A, rectangle.point.second + rectangle.B) << ", ";  
 out << std::pair<U,U> (rectangle.point.first + rectangle.A, rectangle.point.second);  
 out << "]";  
 return out;  
 }  
};  
  
#endif /\* RECTANGLE\_HPP \*/

**Rhombus.h**

#ifndef OOP\_EXERSICE\_03\_RHOMBUS\_H  
#define OOP\_EXERSICE\_03\_RHOMBUS\_H  
#include "Figures.h"  
  
const unsigned int RHOMBUS\_TYPE\_ID = 1;  
  
template<class T1, class T2>  
class Rhombus : public Figures<T1> {  
private:  
 T2 D1, D2;  
public:  
 Rhombus(const std::pair<T1,T2> &\_point, const T2 &d1, const T2 &d2): D1(d1), D2(d2){  
 //point of left edge  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (D1 < 0 || D2 < 0) {  
 throw std::invalid\_argument("Invalid Rhombus parameters!");  
 }  
 }  
  
 void Print(std::ostream& out) override{  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(RHOMBUS\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (T1));  
 file.write((const char\*) &(this->point.second), sizeof(T1));  
 file.write((const char\*) &(this->D1), sizeof(T2));  
 file.write((const char\*) &(this->D2), sizeof (T2));  
 }  
  
 template<class U>  
 friend std::ostream& operator<<(std::ostream& out, const Rhombus<U,U>& rhombus){  
 out << "Rhombus vertices: [ "  
 << std::pair<U,U> (rhombus.point.first, rhombus.point.second) << ", "  
 << std::pair<U,U> (rhombus.point.first + rhombus.D1 / 2, rhombus.point.second + rhombus.D2 / 2) << ", "  
 << std::pair<U,U> (rhombus.point.first + rhombus.D1, rhombus.point.second) << ", "  
 << std::pair<U,U> (rhombus.point.first - rhombus.D1 / 2, rhombus.point.second - rhombus.D2 / 2) << " ]" << std::endl;  
  
 return out;  
 }  
};  
  
#endif //OOP\_EXERSICE\_03\_RHOMBUS\_H

**Trapeze.h**

#ifndef OOP\_EXERSICE\_03\_TRAPEZE\_H  
#define OOP\_EXERSICE\_03\_TRAPEZE\_H  
#include "Figures.h"  
  
const unsigned int TRAPEZE\_TYPE\_ID = 3;  
  
template<class Tpoint, class Tside>  
class Trapeze : public Figures<Tpoint>{  
private:  
 Tside A, B, C;  
public:  
 Trapeze(const std::pair<Tpoint,Tside> &\_point, const Tside &a, const Tside &b, const Tside &c): A(a), B(b), C(c){  
 this->point.first = \_point.first;  
 this->point.second = \_point.second;  
  
 if (A < 0 or B < 0 or C < 0) {  
 throw std::invalid\_argument("Invalid trapeze parameters!");  
 }  
 if (B > A) {  
 std::swap(A, B);  
 }  
 }  
  
 void Print(std::ostream& out) override{  
 out << \*this << std::endl;  
 }  
  
 void Write(std::ofstream& file) override{  
 file.write((const char\*) &(TRAPEZE\_TYPE\_ID), sizeof(unsigned));  
 file.write((const char\*) &(this->point.first), sizeof (Tpoint));  
 file.write((const char\*) &(this->point.second), sizeof(Tpoint));  
 file.write((const char\*) &(this->A), sizeof(Tside));  
 file.write((const char\*) &(this->B), sizeof (Tside));  
 file.write((const char\*) &(this->C), sizeof (Tside));  
 }  
  
 template<class U>  
 friend typename std::enable\_if<is\_double<U,U>::value, std::ostream &>::type  
 operator<<(std::ostream &out, const Trapeze<U,U> &trapeze) {  
 Tside diff = (trapeze.A - trapeze.B) / 2.0;  
 double height = std::sqrt(fabs(trapeze.C \* trapeze.C - diff \* diff));  
 out << "Trapeze vertices: [";  
 out << std::pair<U,U> (trapeze.point.first, trapeze.point.second) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + diff, trapeze.point.second + height) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + trapeze.A - diff, trapeze.point.second + height) << ", ";  
 out << std::pair<U,U> (trapeze.point.first + trapeze.A, trapeze.point.second);  
 out << "]";  
 return out;  
 }  
  
 template<class U>  
 friend typename std::enable\_if<is\_int<U,U>::value, std::ostream &>::type  
 operator<<(std::ostream &out, const Trapeze<U,U> &trapeze) {  
 out << "Trapeze: [ I cant calculate coordinates, because the type is int ]" << std::endl;  
 return out;  
 }  
};  
#endif //OOP\_EXERSICE\_03\_TRAPEZE\_H

1. **Вывод**

В ходе выполнения работы я на практике смог понять как конструировать классы и структуры так, чтобы потом можно было с легкостью добавить функциональность.

**Список литературы**

1. *std::ofstream — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/io/basic_ofstream>(дата обращения 11.12.2020).

1. *std::ifstream — cppreference.com*

URL: <https://en.cppreference.com/w/cpp/io/basic_ifstream>(дата обращения 11.12.2020).